

## User management / directory services

Angermüller Christof

22th May, 2012

# Table of contents

- 1 Configuring slapd
- 2 Adding user information to the LDAP database
- 3 Configuring slapd as a replication provider
- 4 Access control

# Installing the relevant packages

## Command

```
sudo apt-get install slapd ldap-utils libpam-ldap  
libnss-ldap nscd
```

Using following settings in the configuration dialogs:

- dc=tbl // top-level domain component
- cn=admin,dc=tbl // administrator

# Installing the relevant packages

Configuring slapd

Please enter the password for the admin entry in your LDAP directory.

Administrator password:

\*\*\*\*\*

<Ok>

Configuring libnss-ldap

Please choose which account will be used for nss requests with root privileges.

Note: For this to work the account needs permission to access the attributes in the LDAP directory that are related to the users' shadow entries as well as users' and groups' passwords.

LDAP account for root:

cn=admin,dc=tbl

<Ok>

# Reconfiguring slapd

On my Ubuntu machine, I had to reconfigure slapd after installing the relevant packages:

## Command

```
sudo dpkg-reconfigure slapd
```

- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ▶ ↺ 🔍 ↻

# Class attributes

- To find out which attributes are provided by class `posixAccount`, execute

```
ldapvi -h ldapi:/// -b cn=config -Y EXTERNAL
```

- and search for NAME `'posixAccount'`.
- Also look up attributes of superclass SUP

```
474 olcAttributeTypes: {22}( 1.3.6.1.1.1.1.24 NAME 'bootFile' DESC 'Boot image name' EQUALITY caseExactIA5Match SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
475 olcAttributeTypes: {23}( 1.3.6.1.1.1.1.26 NAME 'nisMapName' SUP name )
476 olcAttributeTypes: {24}( 1.3.6.1.1.1.1.27 NAME 'nisMapEntry' EQUALITY caseExactIA5Match SUBSTR caseExactIA5SubstringsMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.26{1024} SINGLE-VALUE )
477 olcObjectClasses: {0}( 1.3.6.1.1.1.2.0 NAME 'posixAccount' DESC 'Abstraction of an account with POSIX attributes' SUP top AUXILIARY MUST ( cn $ uid $ uidNumber $ gidNumber $ homeDirectory ) MAY ( userPassword $ loginShell $ gecos $ description ) )
478 olcObjectClasses: {1}( 1.3.6.1.1.1.2.1 NAME 'shadowAccount' DESC 'Additional attributes for shadow passwords' SUP top AUXILIARY MUST uid MAY ( userPassword $ shadowLastChange $ shadowMin $ shadowMax $ shadowWarning $ shadowInactive $ shadowExpire $ shadowFlag $ description ) )
479 olcObjectClasses: {2}( 1.3.6.1.1.1.2.2 NAME 'posixGroup' DESC 'Abstraction of a group of accounts' SUP top STRUCTURAL MUST ( cn $ gidNumber ) MAY ( userPassword $ memberUid $ description ) )
```

# Obligatory class attributes

```
posixAccount cn, uid, uidNumber, gidNumber  
             homeDirectory
```

```
shadowAccount uid
```

```
inetOrgPerson sn, gidNumber
```

```
    posixGroup cn, gidNumber
```



# Using ldapvi as LDAP client

## Binding as cn=admin,dc=tbl

```
ldapvi -h ldapi:/// --discover -D cn=admin,dc=tbl
```

## Binding as root

- Requires modifying `olcDatabase={1}config.ldif`:  
    `olcAccess: {0}to *`  
    by dn.exact=gidNumber=0+uidNumber=0,  
        cn=peercred,cn=external,cn=auth manage  
    by \* break
- Than following command should also work:  

```
ldapvi -h ldapi:/// --discover
```

# Adding organizational units

- ou people for users:

## Command

```
add ou=people,dc=tbl
objectClass: organizationalUnit
ou: people
```

- ou group for groups:

## Command

```
add ou=group,dc=tbl
objectClass: organizationalUnit
ou: group
```

# Adding a group to ou group

## Command

```
add cn=christof,ou=group,dc=tbl  
objectClass: posixGroup  
cn: christof  
gidNumber: 1008
```

- Using the attributes from /etc/group

# Adding a user to ou people

## Command

```
add uid=christof,ou=people,dc=tbl
objectClass: posixAccount
objectClass: shadowAccount
objectClass: inetOrgPerson
cn: christof
uid: christof
uidNumber: 1008
gidNumber: 1008
homeDirectory: /home/christof
userPassword: {SSHA}7q/EUXHHAG50F5wKl3pZIJ7dbhu5ULOM
sn: Angermueller
```

- Using the attributes from /etc/passwd
- Using slapdpasswd to obtain userPassword

# Database content after adding the components

```
2 ou=people,dc=tbl
objectClass: organizationalUnit
ou: people
```

```
3 ou=group,dc=tbl
objectClass: organizationalUnit
ou: group
```

```
4 cn=christof,ou=group,dc=tbl
objectClass: posixGroup
cn: christof
gidNumber: 1008
```

```
5 uid=christof,ou=people,dc=tbl
objectClass: posixAccount
objectClass: shadowAccount
objectClass: inetOrgPerson
cn: christof
uid: christof
uidNumber: 1008
gidNumber: 1008
homeDirectory: /home/christof
userPassword: {SSHA}7XGDcmphF9lkJI2C5Gfn+fR6M6TVZzi9
sn: christof
```

# Invalidating the name service cache

- It is recommended to invalidate the name service cache:

## Commands

```
nscd -i passwd  
nscd -i group
```

# Editing /etc/nsscd.conf

- Appending **ldap** to the passwd, group and shadow lines:

## /etc/nsswitch.conf

```
passwd:          compat  ldap
group:           compat  ldap
shadow:          compat  ldap

hosts:           files mdns4_minimal [NOTFOUND=return]
networks:        files

protocols:       db files
services:        db files
ethers:          db files
rpc:             db files
```

# Activating LDAP authentication

- Execute `pam-auth-update`
- And choose LDAP Authentication

Pluggable Authentication Modules (PAM) determine how authentication, authorization, and password changing are handled on the system, as well as allowing configuration of additional actions to take when starting user sessions.

Some PAM module packages provide profiles that can be used to automatically adjust the behavior of all PAM-using applications on the system. Please indicate which of these behaviors you wish to enable.

PAM profiles to enable:

```
[*] Unix authentication
[*] LDAP Authentication
[*] GNOME Keyring Daemon - Login keyring management
[*] ConsoleKit Session Management
```

<Ok>

<Cancel>



# Testing the LDAP authentication

## Commands

```
getent passwd  
getent group
```

- These commands now list the added user/group twice
- After commenting out the user/group in
  - /etc/passwd
  - /etc/group
  - /etc/shadow
  - /etc/gshadow
- they are listed only once
- and it should be possible to login with the added user

- 1 Configuring slapd
- 2 Adding user information to the LDAP database
- 3 Configuring slapd as a replication provider
- 4 Access control

# Loading the syncprov module

- For making slapd loading the syncprov module, we have to modify cn=module{0}.ldif:

```
/etc/ldap/slapd.d/cn=config/cn=module{0}.ldif
```

```
dn: cn=module0
objectClass: olcModuleList
cn: module{0}
olcModulePath: /usr/lib/ldap
olcModuleLoad: {0}back_hdb
olcModuleLoad: {1}syncprov
structuralObjectClass: olcModuleList
entryUUID: 887c9b16-3763-1031-9441-8b0f7deefb07
creatorsName: cn=config
```

# Providing `olcDatabase={1}hdb,cn=config`

- ① We create the LDIF file `syncprov.ldif`:

## `syncprov.ldif`

```
dn: olcOverlay=syncprov,olcDatabase={1}hdb,cn=config
changetype: add
objectClass: olcOverlayConfig
objectClass: olcSyncProvConfig
olcOverlay: syncprov
olcSpNoPresent: TRUE
```

- ② which we add by:  
`ldapadd -Y EXTERNAL -H ldapi:/// -f syncprov.ldif`
- ③ Finally, we restart slapd:  
`/etc/init.d/slapd restart`

- 1 Configuring slapd
- 2 Adding user information to the LDAP database
- 3 Configuring slapd as a replication provider
- 4 Access control

# The syntax

## The syntax according to slapd.access

access to <what> by <who> <access> <control>

<what> the item that is accessed

→ \*, dn, attr

<who> the person who wants to access the item

→ \*, dn, self

<access> the permission which is granted

→ none, read, write, manage

<control> the action performed if the rule matches

→ stop, continue, break

# Example 1

```
olcDatabase{0}config.ldif
```

```
olcAccess: {0}to *  
by dn.exact=gidNumber=0+uidNumber=0,  
cn=peercred,cn=external,cn=auth  
manage by * break
```

- Provides root, peercred, external, and auth full access rights to everything

## Example 2

```
olcDatabase{1}hdb.ldif
```

```
olcAccess: {0}  
to attrs=userPassword,shadowLastChange  
by self write  
by anonymous auth  
by dn="cn=admin,dc=tbl" write  
by * none
```

- Authorized users can change their passwords



## Example 3

```
olcDatabase{1}hdb.ldif
```

```
olcAccess: {1}to dn.base="" by * read
```

- Everyone can read everything

## Example 4

```
olcDatabase{1}hdb.ldif
```

```
olcAccess: {2}to *
```

```
by self write
```

```
by dn="cn=admin,dc=tbl" write
```

```
by * read
```

- Everyone can write (and read) its own entries.